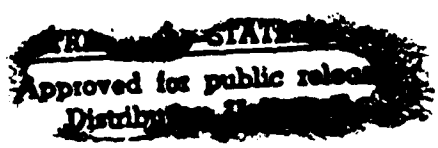
Towards Versatile and Practical Knowledge Acquisition

Yolanda Gil and Cecile Paris
USC/ Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292

June 1993
ISI/RR-93-343



INFORMATION SCIENCES INSTITUTE

310/822-1511
4676 Admiralty Way/Marina del Rey/California 90292-6695

12

Towards Versatile and Practical Knowledge Acquisition

Yolanda Gil and Cecile Paris
USC/ Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292

June 1993
ISI/RR-93-343

In the Proceedings of the IJCAI-93 Workshop on Machine Learning and Knowledge Acquisition, Chambery, France, 1993

REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1993	Research Report

4. TITLE AND SUBTITLE
Towards Versatile and Practical Knowledge Acquisition

5. FUNDING NUMBERS
DABT63-91-C-0025

6. AUTHOR(S)
Yolanda Gil and Cecile L. Paris

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
USC INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292-6695

8. PERFORMING ORGANIZATON REPORT NUMBER
RR-343

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)
ARPA
3701 Fairfax Drive
Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
In the Proceedings of the IJCAI-93 Workshop on Machine Learning and Knowledge Acquisition, Chambery, France, 1993

12A. DISTRIBUTION/AVAILABILITY STATEMENT
UNCLASSIFIED/UNLIMITED

12B. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Rapid prototyping and tool reusability have pushed knowledge acquisition research to investigate method-specific knowledge acquisition (KA) tools appropriate for pre-determined problem-solving methods. We believe that method-dependent KA is not the only approach. The aim of our research is to develop powerful but versatile machine learning mechanisms that can be incorporated into general-purpose but practical knowledge acquisition tools. This paper shows through examples the practical advantages of this approach. In particular, we illustrate how existing knowledge can be used to facilitate KA through analogy mechanisms in a logistics transportation domain and use the same general-purpose mechanism for knowledge acquisition in a different domain through plan generalization. Our hypothetical knowledge acquisition dialogs with a domain expert illustrate which parts of the process are addressed by the human and which parts are automated by the tool, in a synergistic cooperation for knowledge-based extension and refinement. The paper also describes briefly the EXPECT problem-solving architecture and knowledge representation language that facilitate this approach to knowledge acquisition.

14. SUBJECT TERMS
expectation-based knowledge acquisition, knowledge base refinement, knowledge level learning, explanation

15. NUMBER OF PAGES
17

16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reoprts. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month,a nd year, if available (e.g. jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	PR	- Project
G	- Grant	TA	- Task
PE	- Program Element	WU	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the repor.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ...; To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contins classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

Towards Versatile and Practical Knowledge Acquisition

Yolanda Gil and Cécile Paris

USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
U.S.A.
email: gil@isi.edu, paris@isi.edu

In the *Proceedings of the IJCAI-93 Workshop on Machine Learning and Knowledge Acquisition*, Chambery, France, 1993.

Abstract

Rapid prototyping and tool reusability have pushed knowledge acquisition research to investigate method-specific knowledge acquisition (KA) tools appropriate for pre-determined problem-solving methods. We believe that method-dependent KA is not the only approach. The aim of our research is to develop powerful but versatile machine learning mechanisms that can be incorporated into general-purpose but practical knowledge acquisition tools. This paper shows through examples the practical advantages of this approach. In particular, we illustrate how existing knowledge can be used to facilitate KA through analogy mechanisms in a logistics transportation domain and use the same general-purpose mechanism for knowledge acquisition in a different domain through plan generalization. Our hypothetical knowledge acquisition dialogs with a domain expert illustrate which parts of the process are addressed by the human and which parts are automated by the tool, in a synergistic cooperation for knowledge-base extension and refinement. The paper also describes briefly the EXPECT problem-solving architecture and knowledge representation language that facilitate this approach to knowledge acquisition.

Keywords: expectation-based knowledge acquisition; knowledge base refinement; knowledge-level learning; explanation.

1 Introduction

The science of artificial intelligence started with and continues to build on general principles. Newell and Simon's pioneering work on understanding human problem solving shows us that the same mechanisms come to play when trying to find solutions to problems that may seem very different [Newell and Simon, 1972]. GPS implemented this principle and demonstrated that, given domain-specific knowledge, a general-purpose reasoning engine can be used to solve problems in that specific domain, and that the same engine can be used for different problem-solving applications. Decades later, a constellation of expert systems was possible thanks to the expert system shell version of GPS (e.g., [Buchanan and Shortliffe, 1984]). Inference is understood, said common AI wisdom, the hard part is getting the knowledge right. General-purpose inference mechanisms, general-purpose representation mechanisms (a similar argument can be shown with Minsky's frames for the field of knowledge representation) gave AI, like any other science, the beauty of building on principles. Researchers turned then to the next issue in the automated reasoning agenda: machine learning and the acquisition of knowledge. The work ranges from learning heuristic knowledge [Minton, 1988, Knoblock, 1991] to acquiring factual knowledge [Gil, 1992], and shows us that complete automation of the learning or acquisition task in domains of technical expertise is far from reach and that semi-automated general-purpose tools [Davis, 1980] are not enough to achieve efficient prototyping.

At the same time, we witnessed the surge of special-purpose mechanisms to support real-world applications. General-purpose tools seemed incapable of performing well in applications such as manufacturing, robot path planning, or perception. The new common AI wisdom reflects that general-purpose is good, but may not be enough as we currently understand it. The field of knowledge acquisition has rightly taken this lesson. General-purpose KA tools are augmented with application-independent method-specific inference structures [Chandrasekaran, 1986, McDermott, 1988]. Although the resulting tools have less generality, they allow the semi-automated production of fast prototypes in novel domains once the appropriate inference mechanism is manually determined. The inference mechanisms provide expectations about the role of each piece of knowledge that are powerful guidance in the knowledge acquisition process.

However, this approach to building KA tools has limitations that arise from the need of more flexibility than they provide in adapting them to an application [Musen, 1992]. The problem-solving structure of an application cannot always be defined in domain-independent terms. Furthermore, these method-specific inference mechanisms may not address some of the particulars of an application simply because they were designed with generality in mind. Another problem with the method-specific KA tools is that they raise the non-trivial issue of determining a library of possible methods [Chandrasekaran, 1986, McDermott, 1988]. The work involves handcrafting this library of methods making sure to both provide wide-coverage of tasks and well-understood characterizations of the inference capabilities of each method.

To address these limitations, some researchers [Puerta *et al.*, 1992, Klinker *et al.*, 1991] are developing libraries of problem-solving methods that handle finer-grained inference structures than the ones above. These approaches provide much more flexibility in building a KBS, and we share their belief that this is a step in the right direction. Their approach still raises two major issues. One is that the task of building the method libraries still remains. Another issue, which is of major concern in our work, is the accessibility of KA tools to domain experts [Kitto, 1989].

To address the first issue, we advocate the integration of machine learning methods in KA tools that can automatically (or semi-automatically) come up with generalized versions of methods that express domain-specific inference structures and support their reuse in new domains and new situations through analogy. This paper shows through examples the practical advantages of this approach. In particular, we show how to use the same general-purpose mechanisms for knowledge acquisition within a domain and across domains.

The second issue, i.e., accessibility to domain experts, has been central in the design of our architecture [Neches *et al.*, 1985, Swartout *et al.*, 1991, Swartout and Smoliar, 1987]. The problem-solving knowledge and the ability to produce flexible explanations in an interactive dialogue provide the basis for building a KA tool that communicates with an expert much in the way a colleague in their field of expertise would.

The paper runs as follows. We first describe briefly our problem-solving architecture and our knowledge representation language. Then we illustrate how we propose to make use of analogy to facilitate KA with examples from a logistics transportation application that evaluates proposed routings, taking into account restrictions on objects transported, destination points, and vehicles used. Later, we use the same mechanism augmented with generalization to facilitate KA in a different domain for drug prescription. Our hypothetical dialogs with the user illustrate her involvement during knowledge acquisition, as well as which parts of the process are automated by the tool.

2 EXPECT

EXPECT builds upon previous work on the Explainable Expert System (EES) framework [Neches *et al.*, 1985, Swartout *et al.*, 1991, Swartout and Smoliar, 1987], which allows for the construction of expert systems that can provide good explanations of their behavior. In this section, we briefly describe the features of the framework that were found necessary to support good explanations and explain why the same features also allow for the construction of intelligent knowledge acquisition tools.

In EXPECT, different kinds of knowledge are specified in distinct knowledge bases in a high-level specification language. The specific actions that are to be executed by the system to solve a specific problem are *derived* from these knowledge bases by the system, and a record of the derivation is stored to provide the design rationale needed for good explanations. The resulting architecture is shown in Figure 1.

The knowledge bases capture what the system knows about the domain and how to solve problems in that domain. They comprise:

- A domain descriptive knowledge base (or *domain model*), which stores definitions and facts in the domain of the expert system. The domain model is written in the LOOM knowledge representation formalism [MacGregor, 1988].
- A problem-solving knowledge base, which contains an organized collection of plans. The plan language allows for an explicit representation of *intent* (what is to be done to achieve a goal) and supports a wide range of control structures [Project, 1993].

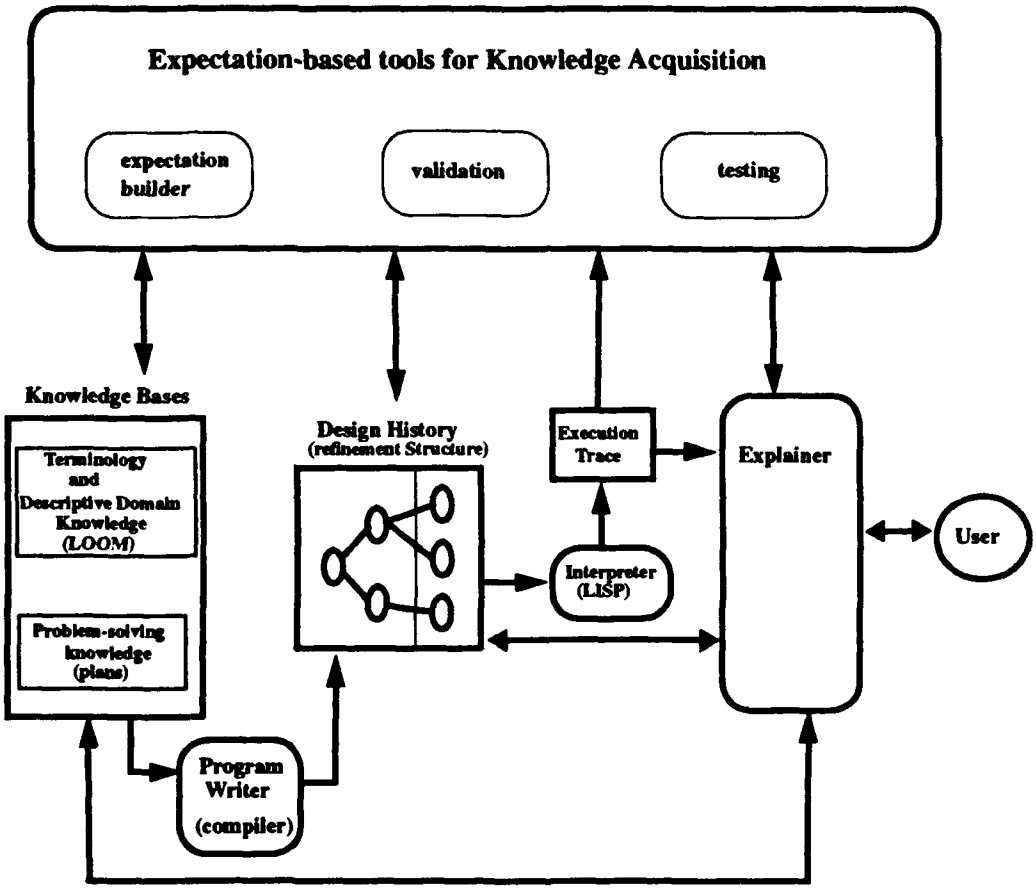


Figure 1: A schematic representation of EXPECT. The design history and the knowledge base components are used to form expectations for knowledge acquisition

```
(defconcept seaport
   :is (:and port
             (:exactly 1 r-location)
             (:exactly 1 r-piers)
             (:exactly 1 r-berth-availability)
             (:exactly 1 r-covered-storage-area)))
```

Figure 2: Definition of a seaport in the domain model

```
(define-plan FIND-IF-SHIP-FITS-IN-SEAPORT
   :capability (determine-whether-fits-in (OBJ (?s is (inst-of ship)))
                                          (IN (?p is (inst-of seaport))))
   :result-type boolean
   :method (less-than (r-ship-length ?s)
                      (compute-max-vessel-length-in-seaport ?p)))


(define-plan COMPUTE-MAX-VESSEL-LENGTH-IN-SEAPORT
   :capability (compute-max-vessel-length-in-seaport
                                    (OBJ (?s is (inst-of seaport))))
   :result-type number
   :method (let ((?berth-types (r-berth-type (r-berth-availability ?s))))
             (max (r-berth-length ?berth-types))))
```

Figure 3: Two plans from the transportation domain.

As an example, consider Figures 2 and 3, which contain samples of these knowledge bases for a logistics transportation domain. In that application, the domain model includes descriptions of `ports`, `seaports`, and `airports`. The representation of a `seaport` is shown in Figure 2: A `seaport` is a type of `port`, and it has attributes such as its `location`, `piers`, `berths`, and `storage areas`.[1]

Figure 3 shows two very simple plans of the problem-solving knowledge base. The capability describes the goals that the plan can achieve, the method represents the body of the plan, and the result type indicates what is returned by the method. The first plan can be used to determine whether a specific type of ship is supported (or "fits in") a given seaport. This is done by testing whether the length of the ship is less than the maximum vessel length allowed in that particular seaport. The second plan finds the maximum ship length a seaport supports based on the length of its berths.

Given a high-level goal, an *automatic program writer* (APW) integrates these structured knowl-

[1] We use the prefix "r-" to indicate names of relations.

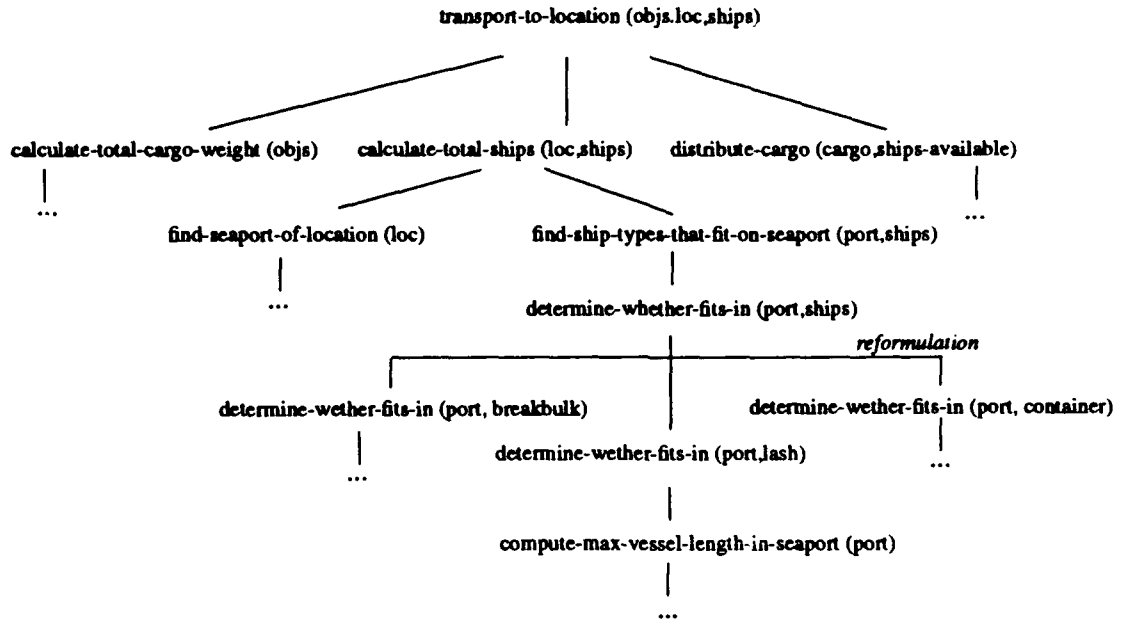


Figure 4: The design history

edge bases by refinement and reformulation from that goal [Neches *et al.*, 1985, Swartout *et al.*, 1991], to produce a system that will be capable of solving specific instances of that goal. The APW records all its steps and decisions in an annotated *design history.* For example, given the general goal to transport a set of objects to a location using a set of ships, the design history produced by the APW is shown in Figure 4. This design history reflects both the goal/subgoal expansion as well as goal reformulations. For example, in Figure 4, we see that the subgoal `determine-whether-fits-in` was reformulated into three goals, one for each type of ship that the system knows of, as indicated in the domain model.

The design history is then available for the system to introspect and explain its reasoning. In order to be able to provide good explanations in an interactive fashion, we have developed a *text planning system* that selects and organizes information from the underlying knowledge sources to construct a coherent text. This text planner supports dialogue with a user [Moore and Paris, 1989, Moore and Paris, 1992]. User input can be stylized English, menu based, or a hypertext-like interaction mousing parts of the explanation to request clarifications [Moore and Swartout, 1990].

As we show in the scenario below, the design history plays a crucial role in the knowledge acquisition process [Paris and Gil, 1993]. In essence, it represents the *functionality of the knowledge that the domain model contains.* It thus allows the system to reason about *how* knowledge will be used. This is crucial as one cannot indiscriminately add new knowledge into a system but rather needs to understand how that knowledge will be used. Otherwise, there is a danger that the knowledge added be useless or incomplete to achieve the task for which the system is designed.

Because the system includes an explanation facility capable of producing coherent explanations of the system's behavior, the KA tool can justify to the user why it is asking specific questions and provide feedback as to what is being added or changed. Furthermore, the user can then at any point request documentation about any part of the knowledge base.

3 Transfer of Problem-Solving Knowledge Within a Domain

Figure 5 shows a scenario in which existing problem-solving knowledge is corrected. The system is given the problem of transporting a unit to the seaport of Cabra (line [1]). The system solves the problem and reaches the conclusion that it takes 3 days (line [2]). The user is surprised that it could be done so fast in such a small seaport and asks why (line [3]). The system explains its conclusion by retracing its reasoning (line [4]), exploiting both the actual execution trace for this specific problem and the design history. Notice that, at this point, only a summary of the whole reasoning is included in the explanation, thus providing a high-level justification for the conclusion. The user can however "zoom-in" on particular part of the problem solving by asking further questions. This is illustrated in lines [5-6].

The justifications provided by the system allow the user to detect a potential error: he or she disagrees with some of the information given by the system and provides conflicting information (line [7]). Given this new information, EXPECT now reasons about its execution trace, attempting to localize the problem, and a dialogue to debug the system is initiated.

First, EXPECT finds the exact point in the execution trace that led to the conclusion questioned by the user and explains that part of the reasoning (line [8]). This allows the user to detect the problem more precisely. In this case, the user realizes that the plan to determine whether a ship fits in a port is incorrect (or incomplete). The user thus provides the additional information, namely that the draft of a ship must also be taken into consideration (line [9]). The system asks the user how to check the draft. Because the system already has a plan to check the length, it uses it to form an expectation for how to check the draft, and presents it as a suggestion to the user (line [10]). Upon positive reaction (line [11]), EXPECT prepares the elements that can be the source of the analogy, and prompts the user for their correspondents (line [12]). After the user's response (line [13]), EXPECT summarizes to the user the changes made to the knowledge base (line [14]). Notice that since the results of both checks are boolean, the system also has a suggestion about how to combine them. The plan is now corrected. After the problem has been fixed, EXPECT re-works the original goal (line [16]).

Figure 6 shows a different scenario where new plans are added to the knowledge base by the user. Much like in the previous scenario, the user communicates to the system the analogy of ships and ports with aircrafts and hangars: what to retrieve (line [2]), what the correspondences are (line [4]), and what the exceptions are (lines [6-10]). The system takes charge of the rest, including updating the corresponding parts of the different knowledge sources. Notice that since the generation system uses concrete items of the knowledge base to generate components of the sentence, it is easy for EXPECT to reverse the process and identify the correspondences in the analogy proposed by the user.

U:	How long does it take to transport that unit to Cabra with the ships available?	[1]
S:	3 days	[2]
U:	Why?	[3]
S:	To transport unit X to a location, first I calculated that the total weight of the cargo is 300,000 tons, then I determined that I could use 3 container ships and 10 breakbulk, and distributed the cargo among the ships.	[4]
U:	Why breakbulk ships?	[5]
S:	I used the Cabra seaport, and determined that breakbulk ships fit in the Cabra seaport.	[6]
U:	This is wrong. Cabra is too small for breakbulk ships.	[7]
S:	I concluded that breakbulk ships fit in the Cabra seaport. To do this, I checked that the ship length is less than the maximum vessel length in that seaport. Is this right? (Yes; No; Yes with constraints)	[8]
U:	Yes, with constraints: you have to check both the length and the draft.	[9]
S:	I know how to check the length. Do I check the draft in a similar way? (Yes; No; Yes with constraints)	[10]
U:	Yes	[11]
S:	What corresponds to (1) the ship length (2) less than (3) the maximum vessel length in that seaport?	[12]
U:	(1) ship draft (2) less than (3) maximum vessel draft in the seaport	[13]
S:	Ok. So to determine what ship types fit in a seaport, I determine if the ship length is less than the maximum vessel length in that seaport, and then I determine if the ship draft is less than the maximum vessel draft allowed in the seaport. Both have to be true in order for a ship type to fit in a seaport. Correct? (Yes or No)	[14]
U:	Yes	[15]
	[...]	
S:	Re-solving the problem with the updated method for determining what ship types fit in a seaport. It takes 10 days to transport the unit to Cabra.	[16]
	⋮	

Figure 5: Hypothetical scenario – Modifying existing plans

S:	...	[1]
U:	To service a type of aircraft, first check if it fits in a hangar. Similar to how a ship fits in a seaport.	[2]
S:	To determine what ship types fit in a seaport, I determine if the ship length is less than the maximum vessel length in that seaport, and then I determine if the ship draft is less than the maximum vessel draft allowed in the seaport. Both have to be true in order for a ship type to fit in a seaport. We want to describe a way to determine if a type of aircraft fits in a hangar. What corresponds to (1) the ship length (2) less than (3) the maximum vessel length in that seaport (4) the ship draft (5) less than (6) the maximum vessel draft allowed in the seaport (7) both have to be true?	[3]
U:	(1) aircraft length (2) less than (3) hangar length (4) aircraft height (5) less than (6) hangar height (7) both have to be true.	[4]
S:	Ok. So in order to check if a type of aircraft fits in the hangar, I determine if the aircraft length is less than the hangar length, and then I determine if the aircraft height is less than the hangar height. Both have to be true in order for an aircraft type to fit in the hangar. [Yes; No; Yes with constraints]	[5]
U:	Yes, with constraints: you have to check length, height, and width.	[6]
S:	Ok. I know the length, width, and height of hangars. I know the length and height of an aircraft. What is the width of an aircraft?	[7]
U:	Wing span?	[8]
S:	I understand. I know how to check the length and the height. Do I check the width in a similar way? (Yes; No; Yes with constraints)	[9]
U:	Yes	[10]
S:	Ok. So in order to check if a type of aircraft fits in a hangar ...	[11]

Figure 6: Hypothetical scenario – Adding new plans

4 Transfer of Problem-Solving Knowledge Across Domains

The analogies in the above scenarios involve plans within the same domain, but exactly the same mechanism would work if the new plans were in a different domain. For example, the user could suggest the similarities with checking if a package fits in a truck, or if any physical object fits in another. Because the analogy is not worked out automatically, but instead the user is in charge of guiding the KA tool through all the stages (retrieval, mapping, and adaptation), the analogy can be as far fetched as the user finds suitable. For example, one can imagine using the plan to check if a ship fits in a port to check if an event happens during another one by checking if the corresponding time intervals fit within one another. Thus, the same general mechanism can be used throughout any stage of knowledge acquisition and throughout any application domain at hand. Notice that this requires that knowledge acquisition does not happen in a vacuum, and by that we mean that the system should have access to as many knowledge bases in different domains as possible. Some efforts already aim in this direction [Neches *et al.*, 1991, Lenat and Guha, 1989]. This is much like humans communicate and learn things from each other: by first establishing backgrounds and then using constructs that the other person is accustomed to. For example, doctors may not teach new things to biologists in the same way that they would explain them to engineers, since they would have common ground concepts with the former to build upon.

Of course, acquiring knowledge by analogy has its limits. For example, a user who is entering knowledge in a medical domain may not know in detail what problem solving knowledge is involved in our transportation domain. This is also common in humans: the more we know about the other person's areas of knowledge, the easier it is to explain things to them. Again, we can turn to machine learning techniques for an answer. Let us consider an example from our transportation domain. There is a plan to calculate the throughput (takeoffs per day) that an airport can handle. To do that, the given throughput is adjusted by a percentage factor determined by the presence of bad weather, and by how many docks are available for unloading.[2] There is also a different plan to calculate how many planes are available for a transportation problem. To do so, the amount of planes assigned for the task is adjusted by a percentage factor that takes into account the rate at which each type of plane tends to break down and how much time it takes to repair it. Induction techniques, and in particular, generalization from examples could be applied in these cases to provide the system with the concept of a plan to do adjustments of values, as shown in Figure 7. We believe that the type of language that we use in EXPECT's plans combined with the Loom classifier would facilitate this, but in addition we plan to research on general-purpose machine learning methods to induce these generalized plans automatically or semi-automatically. In the later case, a knowledge engineer would be involved, but we believe this is a much lighter burden than she would have in current KA environments. Either way, we believe that these machine learning methods would be much more useful for automating KA than a general KA tool for a particular generic method.

The following example illustrates how these generalized plans could be used to cooperate with an expert during KA. Suppose that a medical expert is using EXPECT for administering drugs, in particular to advice with digitalis therapy [Swartout, 1983]. The default dose of a drug has to be reduced if the patient has certain conditions (called sensitivities to the drug). To treat a patient with

[2] If there are not enough unloading docks the planes would be idle waiting for a dock to become available.

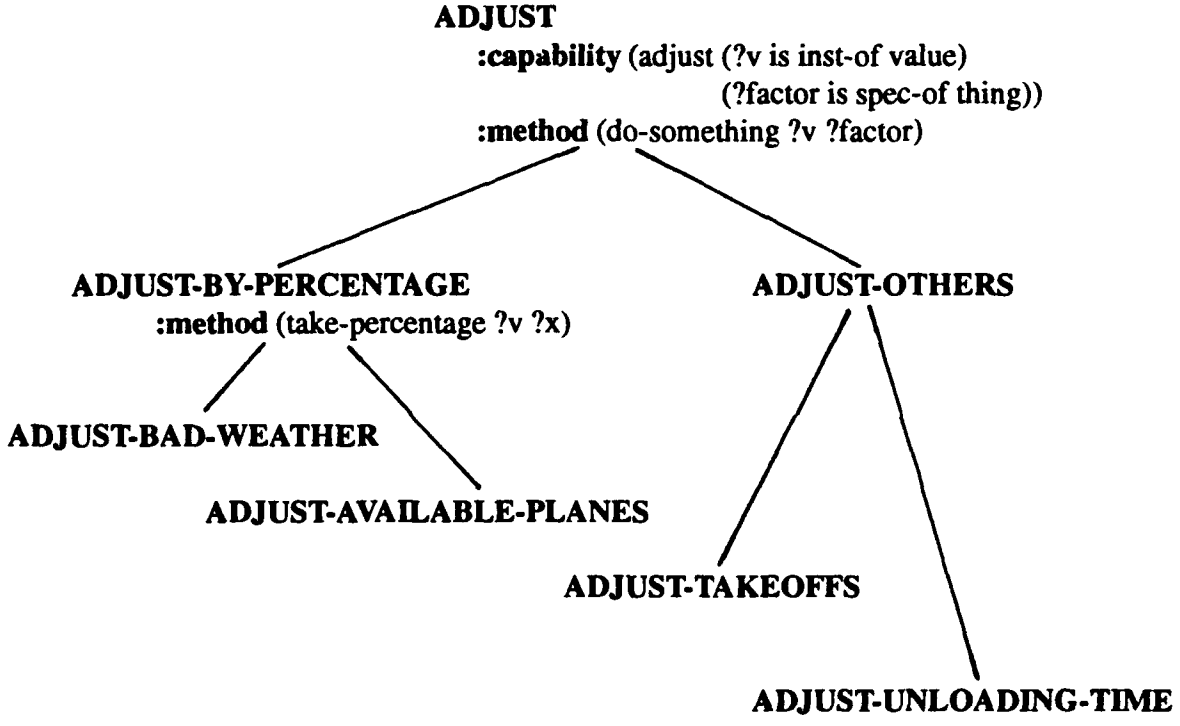


Figure 7: A hierarchy of plans for adjusting values

digitalis, the normal dose (1 μg/Kg) needs to be adjusted by a factor of 0.8 if the patient has a high level of serum calcium and by a factor of 0.7 when serum potassium is low. If both sensitivities are present, the default dose is adjusted by the product of both factors (0.8 * 0.7). This example is used in our next scenario.

Suppose that the system knows only about one sensitivity (low serum potassium), as well as how to adjust the dose of digitalis if the patient has this condition. Figure 8 shows a scenario where the user wants to add a new sensitivity: high serum calcium. The user enters whatever information she is aware that the system needs about drug sensitivities in line [1]. EXPECT updates the domain model accordingly. Now EXPECT examines the problem-solving knowledge to determine what are sensitivities used for. This causes the question in line [2] of how to adjust the dose for this new sensitivity. The user can ask questions at any time. Lines [3-6] show how the system explains to the user the reasoning behind this knowledge acquisition episode, and cooperate with the user until she understands the reason for this question. The design history is used in this case to produce explanations of why and how drug doses are adjusted for sensitivities. In line [9] the descriptive domain knowledge is used this time to address the user's question. Notice that the explanation module is using all the different sources of information and presenting justifications at the right level of abstraction or detail, but it is completely transparent to the user what source is being used. We take advantage of this ability for knowledge acquisition: the system updates whichever source of knowledge needs to be updated, and the user is unaware of where and how changes are made.

The system knows to acquire an adjustment for this sensitivity based on how sensitivities are used for problem solving. Because a general plan for doing adjustments is available the system knows that when more than one adjustment is present it needs a method to recombine the results (line[12]). The system can justify the reason for the questions to the user (lines [13-14]), as well as to produce examples that undo the generalization and facilitate a ground analogy to the user in terms of airport takeoffs (line [16]).

In summary, this scenario illustrates how EXPECT uses plan generalization and analogical reasoning to guide knowledge acquisition. The knowledge about adjustments in the transportation domain is used to acquire adjustments for drug therapy. We used two very different domains to illustrate the potential of our approach, but the impact should be greater when the nature of the domains is not so diverse.

5 Related Work

NeoDISCIPLE [Tecuci, 1992] integrates several machine learning techniques in a knowledge acquisition tool. NeoDISCIPLE takes a user-given answer to a problem and applies explanation-based learning to build a plausible proof tree, abduction to complete the proof, and several other learning techniques to generalize the proof. Its predecessor, DISCIPLE [Tecuci and Kodratoff, 1990], built an analogy with an existing proof when the system lacked domain knowledge to build the proof for a new input. Our approach automates different parts of the process. EXPECT provides a framework for analogical reasoning where the tool takes responsibility for suggesting corrections and for making the adequate changes in the knowledge base. The user suggests the source of the analogies, the mapping, and any necessary adaptations. The tool provides the supporting environment for navigating through the system's reasoning and carrying out the user's corrections through analogical reasoning or any other mechanisms.

U:	Add a new sensitivity for digitalis: high serum calcium. [...]	[1]
S:	Ok. What method should I use to adjust the dose for high serum calcium sensitivity [Take percentage; Do computation; Other]?	[2]
U:	Why?	[3]
S:	I know another sensitivity for digitalis: low serum potassium. I always adjust the dose of digitalis for low serum potassium.	[4]
U:	Why?	[5]
S:	In order to administer digitalis, I need to anticipate toxicity. To anticipate toxicity, I calculate the normal dose and then I adjust it according to sensitivities. Which method should I use to adjust [Take percentage; Do computation; Other]?	[6]
U:	Take percentage	[7]
S:	Ok. Which percentage?	[8]
U:	What is the normal dose?	[9]
S:	1μg/Kg. Which percentage?	[10]
U:	0.8	[11]
S:	Ok. Now I need to know how to combine the results of the adjustments. Choose one of [MIN;MAX;ADD;MULTIPLY;Other].	[12]
U:	Why?	[13]
S:	We are adjusting digitalis dose. To do an adjustment, I need to know: (1) value to be adjusted, (2) factors, (3) methods to adjust the value, and (4) how to combine the results. In this case, (1) digitalis dose (2) drug sensitivities (3) take percentage 0.8 for high serum calcium, take percentage 0.7 for low serum potassium. Now I need to know (4) how to combine the results.	[14]
U:	How to combine the results?	[15]
S:	For example, when I compute the takeoffs per day in an airport, I adjust the default value of takeoffs for the airport according to weather conditions and to available unloading docks. To do an adjustment, I need to know: (1) value to be adjusted, (2) factors, (3) methods to adjust the value, and (4) how to combine the results. In this case, (1) takeoffs per day (2) weather, unloading docks (3) take percentage, compute docks throughput (4) take minimum value. In the digitalis dose adjustment, I need to know how to combine the results of the adjustments. Choose one of [MIN;MAX;ADD;MULTIPLY;Other].	[16]
U:	multiply	[17]
S:	Ok. High serum calcium is now a new sensitivity for digitalis.	[18]

Figure 8: Hypothetical scenario – Adding new plans in a different domain

The Spark/Burn/FireFighter framework [Klinker *et al.*, 1991] treats knowledge acquisition as a programming effort, and aims to provide a set of *mechanisms* as basic blocks for building knowledge-based systems. The goal is to design such mechanisms to be both *usable* (understandable by domain experts) and *reusable* (applicable to several tasks and domains). The PROTEGE-II system [Puerta *et al.*, 1992] is also based on reusable mechanisms as building blocks for knowledge acquisition, and addresses usability by integrating domain-dependent knowledge in the process of putting these mechanisms together. EXPECT shares both goals, and the aim is to achieve them without explicitly building such mechanisms. If at all, these mechanisms would be a by-product of the tool's interaction with the user and its learning capabilities. Problem-solving knowledge in EXPECT is usable because it is accessible to domain experts through explanations. Knowledge is reused when the user proposes analogies with new situations, as well as through any generalizations over existing cases.

Another general-purpose knowledge acquisition approach is that proposed by KADS [Wielinga *et al.*, 1992], a methodology for building knowledge based systems, which, unlike EXPECT, emphasizes the initial building of a system (as opposed to its refinement). KADS views knowledge acquisition as a modelling activity and proposes a number of models that a knowledge engineer needs to build. Each model emphasizes a specific aspect of the system to construct and contains part of the knowledge needed. KADS also separates the domain model from the control knowledge, which is in turn divided into the inference model, the task model, and the strategic knowledge. These last three models together indicate how knowledge from the domain model is to be used in problem solving, and could thus theoretically be also used to guide refinement of a KBS once an initial prototype has been built. This is not currently done, however. KADS's current aim is to help the knowledge engineer faced with the task of building a new KBS by providing an explicit set of building blocks (or models) that the knowledge engineer should be concerned about, thus using a divide-and-conquer approach to the initial knowledge acquisition task. Our aim in EXPECT is a complementary one: it is to aid in refining and debugging a KBS automatically once a prototype system exist. Because EXPECT derives code from the specification automatically (i.e., the domain model and the plans), changes done at the specification level are automatically reflected in the executable code. In contrast, until executable systems are automatically derived from the models designed with the KADS methodology, guidance from the models would only apply to refining the models themselves. The changes would then need to be reflected in the code by hand.

6 Conclusion and Future Work

We have shown our approach to building KA tools to achieve fast prototyping and knowledge re-use based on general-purpose machine learning methods. In essence, the current content of the knowledge base is the basis for dynamically creating expectations for what knowledge is to be acquired. The contents of knowledge bases that corresponds to other applications facilitate KA as well, through transferring general problem solving knowledge shared by different areas of expertise. We believe that the type of interactions with the user shown in our scenarios make the tool more accessible to domain experts than other approaches. Ultimately, we would like to present experts with a KA tool that has learning abilities as well as some basic knowledge about the field.

Acknowledgments

We would like to thank Kevin Knight, Vibhu Mittal, Bill Swartout, and the anonymous reviewers for their helpful comments on earlier drafts of this paper. The research is possible thanks to the efforts of current and previous members of the EES group. We gratefully acknowledge the support of ARPA with the contract DABT63-91-C-0025.

References

[Buchanan and Shortliffe, 1984] Bruce G. Buchanan and Edward H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley Publishing Company, 1984.

[Chandrasekaran, 1986] B. Chandrasekaran. Generic tasks in knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, 1(3):23–30, Fall 1986.

[Davis, 1980] Randall Davis. *Knowledge-based systems in artificial intelligence.* McGraw-Hill, 1980.

[Gil, 1992] Yolanda Gil. *Acquiring Domain Knowledge for Planning by Experimentation.* PhD thesis, Carnegie Mellon University, School of Computer Science, 1992.

[Kitto, 1989] C.M. Kitto. Progress in Automated Acquisition Tools: How close are we to replacing the knowledge engineer. in *Knowledge Acquisition*, Volume 1, 1989, 1989.

[Klinker *et al.*, 1991] Georg Klinker, Carlos Bhola, Geoffroy Dallemagne, David Marques, and John McDermott. Usable and Reusable Programming Constructs. *Knowledge Acquisition*, 3(2):117–135, 1991.

[Knoblock, 1991] Craig A. Knoblock. *Automatically Generating Abstractions for Problem Solving.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1991.

[Lenat and Guha, 1989] Douglas B. Lenat and R. V. Guha. *Building Large Knowledge-Based Systems.* Addison-Wesley Publishing Company., Reading, Massachusetts, 1989.

[MacGregor, 1988] Robert MacGregor. A Deductive Pattern Matcher. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, Minnesota, August 1988.

[McDermott, 1988] John McDermott. Preliminary steps towards a taxonomy of problem-solving methods. In *Automating Knowledge Acquisition for KBS.* Kluwer, 1988.

[Minton, 1988] Steve Minton. *Learning Search Control Knowledge: An Explanation-based Approach.* Kluwer Academic Publishers, Boston, Massachusetts, 1988.

[Moore and Paris, 1989] Johanna D. Moore and Cécile L. Paris. Planning Text For Advisory Dialogues. In *Proceedings of the Twenty-Seventh Annual Meeting of the Association for Computational Linguistics*, pages 203–211, Vancouver, B.C., Canada, June 26-29 1989.

[Moore and Paris, 1992] Johanna D. Moore and Cécile L. Paris. Planning Text for Advisory Dialogues: Capturing Intentional, Rhetorical and Attentional Information, 1992. Technical Report from the University of Pittsburgh, Department of Computer Science (Number 92–22) and from USC/ISI, # RS 93-330.

[Moore and Swartout, 1990] Johanna D. Moore and William R. Swartout. Pointing: A Way Toward Explanation Dialogue. In *Proceedings of the National Conference on Artificial Intelligence*, pages 457–464, Boston, MA, July 29 - August 3 1990.

[Musen, 1992] Mark A. Musen. Overcoming the Limitations of Role-Limiting Methods. *Knowledge Acquisition*, 4(2):165–170, 1992.

[Neches *et al.*, 1985] Robert Neches, William R. Swartout, and Johanna D. Moore. Enhanced Maintenance and Explanation of Expert Systems Through Explicit Models of Their Development. *IEEE Transactions on Software Engineering*, SE-11(11):1337–1351, November 1985.

[Neches *et al.*, 1991] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, and W. R. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56, 1991.

[Newell and Simon, 1972] Allen Newell and Herbert A. Simon. *Human Problem Solving*. Prentice-Hall, New Jersey, 1972.

[Paris and Gil, 1993] Cécile L. Paris and Yolanda Gil. EXPECT: Intelligent support for knowledge-based system development. In *Proceedings of the Seventh European Knowledge Acquisition for Knowledge-Based Systems Workshop*, Toulouse, France, September 1993. To be published by Springer Verlag, in their Lecture Notes in Computer Science Series.

[Project, 1993] The EES Project. The Explainable Expert System: user's manual, 1993. Working notes.

[Puerta *et al.*, 1992] Angel R. Puerta, John W. Egar, Samson W. Tu, and Mark A Musen. A Multiple-Method Knowledge-Acquisition Shell for the Automatic Generation of Knowledge-Acquisition Tools. *Knowledge Acquisition*, 4(2):171–196, 1992.

[Swartout and Smoliar, 1987] William R. Swartout and Stephen W. Smoliar. On Making Expert Systems More Like Experts. *Expert Systems*, 4(3):196–207, August 1987.

[Swartout *et al.*, 1991] William R. Swartout, Cécile L. Paris, and Johanna D. Moore. Design for Explainable Expert Systems. *IEEE Expert*, 6(3):58–64, June 1991.

[Swartout, 1983] William R. Swartout. XPLAIN: A system for creating and explaining expert consulting systems. *Artificial Intelligence*, 21(3):285–325, September 1983. Also available as ISI/RS-83-4.

[Tecuci and Kodratoff, 1990] G. Tecuci and Y. Kodratoff. Apprenticeship Learning in Imperfect Domain Theories. In *Machine Learning: An Artificial Intelligence Approach*, volume 3. Morgan Kaufmann, San Mateo, CA, 1990.

[Tecuci, 1992] Gheorghe D. Tecuci. Automating Knowledge Acquisition as Extending, Updating, and Improving a Knowledge Base. *IEEE transactions on Systems, Man, and Cybernetics*, 22(6):1444–1460, 1992.

[Wielinga *et al.*, 1992] B.J. Wielinga, A. Th. Schreiber, and A. Breuker. KADS: a modelling approach to knowledge acquisition. *Knowledge Acquisition*, 4(1):5–54, 1992.